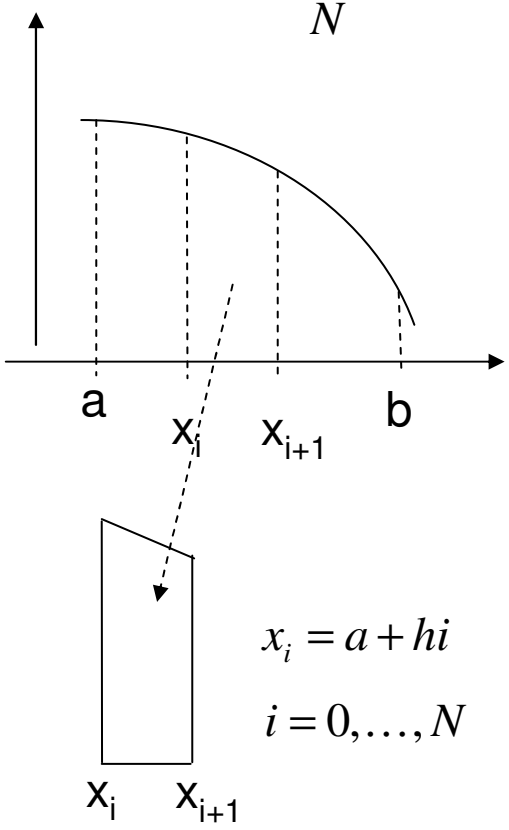
Estructuras de datos

Ejemplo: Cálculo de la integral de un polinomio en un intervalo [a,b] dado.

N: número de subintervalos en [a,b]

$$h = \frac{b-a}{N}$$



$$x_i = a + hi$$

$$i = 0, \ldots, N$$

$$area = \frac{f(x_i) + f(x_{i+1})}{2} h$$

$$area\ \ total = h \sum_{i=0}^{N-1} \frac{f(x_i) + f(x_{i+1})}{2} =$$

$$\frac{h}{2}(f(x_0) + 2f(x_1) + \cdots + 2f(x_{N-1}) + f(x_N)) =$$

$$h\left\{ \left(\frac{f(a) + f(b)}{2}\right) + \sum_{i=1}^{N-1} f(x_i) \right\}$$

```
1  % Se aproxima el valor de la integral de una
2  % funcion f en el intervalo [a,b] mediante
3  % el metodo del trapecio
4  if a > b
5      temp = a;
6      a = b;
7      b = temp;
8  end
9  N = 1.e5;
10 h = (b-a)/N;
11 v = a + (1:N-1)*h;
12 int = h * ( (f(a)+f(b))/2 + sum(f(v)) );
```

archivo : calculo_integral.m

Estructuras de datos

Cadenas de caracteres

A menudo es necesario manipular texto en los programas

- Declaración de un texto

  >> t = ‘esto es un texto’

- Acceso a las entradas de una cadena de caracteres

  >> d = t(1:9) → almacena en la variable d el texto: ‘esto es u’

  de la posición 1 a la 9

  (incluye los espacios en blanco)

  >> d = t(3:10) → almacena en la variable d el texto: ‘to es un’

  de la posición 3 a la 10

Estructuras de datos

Cadena de caracteres

- Operadores sobre cadenas de caracteres

  Formas de concatenar texto: strcat y strvcat

  >> a = ‘pepe’

  Concatenación horizontal:

  >> strcat(‘yo me llamo ‘, a, ‘ blanco’) → almacena en la variable

  ans el texto: yo me llamo pepe blanco

  Concatenación vertical:

  >> strvcat(‘yo me llamo ‘, a, ‘ blanco’) → almacena en la variable

  ans el texto: yo me llamo

  pepe

  blanco

  >> t = [‘yo me llamo’ blanks(1) a blanks(1) ‘blanco’]

  Almacena en la variable t el texto: yo me llamo pepe blanco

  blanks(1) asigna un espacio en blanco

Estructuras de datos

Cadena de caracteres

- Operadores char y abs

  s = char(x): convierte un entero x (código ascii) a caracter (texto) y lo almacena en la variable s

  x = abs(s): convierte el caracter s (texto) a su código ascii y lo almacena en la variable x

  >> abs('a') → 97

  >> char(97) → a

  >> abs('saul') → 115 97 117 108

  >> char([115 97 117 108]) → saul

  >> char(32) → (espacio en blanco)

  >> abs(' ') → 32

- Funciones "ischar" y "isnumeric"

  ischar(s) retorna 1 si s es una cadena, en cualquier otro caso 0.

  >> ischar('11') → 1

Estructuras de control

Ejemplo: uso del “if” múltiple anidado y manejo de cadenas de caracteres

genera el vector “genero” con entradas “f” y “m”.

```
1  % generacion de los vectores genero, edad e hijos
2  % estos son almacenados en el archivo 'gen_edad_hijos'
3  clear
4  for i=1:1000
5      if (mod(i,2)==0), genero(i)='f';
6      elseif (mod(i,3)==0), genero(i)='m';
7      elseif (mod(i,5)==0), genero(i)='m';
8      elseif (mod(i,7)==0), genero(i)='m';
9      else
10         genero(i)='f';
11     end
12 end
13 genero = genero';
14 edad = 14 + round(57*rand(1000,1));
15 hijos = round(10*rand(1000,1));
16 save gen_edad_hijos genero edad hijos;
```

archivo : gen_edad_hijos.m

¿Como calcular el número de ‘f’ y ‘m’ en el vector genero?

```
>> vectores_aleatorios
>> load gen_edad_hijos
>> mf = ‘m’-’f’;
cantidad de ‘m’:
>> sum(genero - 'f')/mf
ans = 272
cantidad de ‘f’
>> - sum(genero - 'm')/mf
ans = 728
```

nota: funciones “double”, “char” y “abs”

tabla ascii

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Estructuras de datos

Cadenas de caracteres

- Operadores num2str y str2num permiten pasar un valor numérico a caracter y viceversa

  >> t = num2str(pi,3) → corresponde al texto: 3.14 (3 dígitos)

  y es almacenado en la variable t

  >> a = [‘el número pi es’ blanks(1) t] → concatena y almacena

  en a el texto: el número pi es 3.14

  >> x = str2num(t) → corresponde al número: 3.14

>> S = ['1 2'; '3 4']

>> str2num(S)

```
>> S = ['1 2'; '3 4']
S =
1 2
3 4
>> str2num(S)
ans =
     1     2
     3     4
```

Estructuras de datos

Cadenas de caracteres

- Operador eval permite evaluar una expresión válida en MATLAB pero escrita en forma de texto

  >> x = eval(‘2 + 4’) → interpreta el texto, lo ejecuta como instrucción válida en MATLAB y lo almacena en la variable x

  >> a = eval(‘sin(pi)’) → almacena en a el número 1.2246e-016

  >> for n = 1:4; eval(['M' num2str(n) ' = magic(n) ']); end

  genera una sucesión de matrices con nombres M1 a M4

M1 =

```
     1
```

M2 =

```
     1     3
     4     2
```

M3 =

```
     8     1     6
     3     5     7
     4     9     2
```

M4 =

```
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Estructuras de datos

Cadenas de caracteres

- Las funciones “length”, “size” y “find” sobre una cadena de caracteres:

  >> a = 'esto es una prueba‘

  >> length(a) retorna el número de caracteres de la cadena, es decir 18

  >> find(a == ‘e’) retorna el vector 1 6 16, correspondientes a las posiciones de la letra “e” sobre el arreglo “a”

  >> S = ['1 2'; '3 4'];

  >> size(S) → 2 3

  size(S) retorna el número de filas y columnas de la “matriz”.

  >> find(S == '4') → 6

- Función “strcmp” permite comparar cadenas de caracteres, así

  strcmp(s1,s2) retorna 1 si s1 y s2 son iguales, si no retorna 0.

  >> s1 = ‘sss’; s2 = ‘ss’; a = strcmp(s1,s2) → a = 0

  >> s1 = ‘sss’; s2 = ‘sss’; a = strcmp(s1,s2) → a = 1

Estructuras de datos

Cadena de caracteres

- Función “findstr” permite encontrar una cadena de caracteres sobre otra cadena dada.

  findstr(s1,s2) retorna las posiciones (índice) del primer carácter de la cadena más corta sobre la cadena más larga.

  >> s = ‘Como estas y donde estas?’;

  >> a = findstr(s,’m’) → a = 3

  >> a = findstr(s,’estas’) → a = [6 20]

  >> a = findstr(s,’Estas’) → a = [ ]

  >> a = findstr(s,’ ’) → a = [5 11 13 19]

- Función “inline”: construye una función a partir de una cadena de caracteres,

  Por ejemplo f = inline('cos(x)+2*sin(x) ')

  y así poder calcular f(pi/3) → 2.2321

Estructuras de datos

Ejemplo: construcción de la fórmula de un polinomio dado sus coeficientes en orden decreciente de las potencias como un vector

```
% construucion de la formula de un polinomio dado
% sus coeficientes en orden decreciente de las
% potencias como un vector
p5 = input('ingrese los coeficientes del polinomio  ');
n = length(p5);
pp = [];
for i = 1:n
    ss = '+';
    if sign(p5(i)) == -1
        ss = '-';
    end
    pp = [pp ss num2str(abs(p5(i)),8) '*x.^' num2str(n-i,1)];
end
f = inline(pp);
disp(f);
```

archivo : const_formula_polinomio.m

Estructuras de control

Instrucciones de entrada (lectura) y de salida (escritura) (cont.)

A continuación veremos funciones para lectura y escritura de archivos.

- Funciones fopen y fclose

  Estas funciones sirven para abrir y cerrar archivos, respectivamente.

  La función fopen tiene la forma:

  [fi, texto] = fopen(nom_archivo, c)

  donde **fi** es una variable que recibe el valor de retorno que identifica al archivo de nombre nom_archivo, **texto** es un mensaje para el caso de que se produzca un error, y **c** es un carácter (o dos) que indica el tipo de operación que se desea realizar con el archivo. Las opciones más importantes para **c** son las siguientes:

  'r' lectura (de read)

  'w' escritura reemplazando (de write)

  'a' escritura a continuación (de append)

  'r+' lectura y escritura

Estructuras de control

Instrucciones de entrada (lectura) y de salida (escritura) (cont.)

- Funciones fopen y fclose (cont.)

  Cuando por alguna razón el archivo no puede ser abierto, se devuelve un -1 en la variable fi. En este caso el valor de retorno texto puede proporcionar información sobre el tipo de error que se ha producido.

  Después de realizar las operaciones de lectura y escritura deseadas, el archivo se puede cerrar con la función fclose en la forma siguiente:

  st = fclose(fi)

  donde st es un valor de retorno para posibles condiciones de error, (cuando st retorna como 0 significa que el archivo cerró bien).
  Si se quieren cerrar a la vez todos los archivos abiertos puede utilizarse el comando:

  st = fclose('all')

Estructuras de control

Instrucciones de entrada (lectura) y de salida (escritura) (cont.)

- Funciones fscanf y fprintf

  Estas funciones permiten leer y escribir en archivos ascii, es decir, en archivos formateados.

  Forma general de la función fscanf:

  [var, count] = fscanf(fi, format, size)

  Lee del archivo identificado por **fi** de acuerdo a un formato especificado por **format** y lo devuelve en la variable **var** (real, vector o matriz).

  count (opcional) devuelve el número de elementos leídos satisfactoriamente.

  size (opcional) limita el número de elementos a ser leidos en el archivo Si no se coloca, lee hasta el final del archivo. Si se especifica, se tienen las siguientes opciones

N	lee a lo más N elementos en un vector columna
inf	lee hasta el final del archivo
[M,N]	lee a lo más M * N elementos llenando al menos una matriz M×N siguiendo el orden de columnas. N puede ser inf, pero no M.

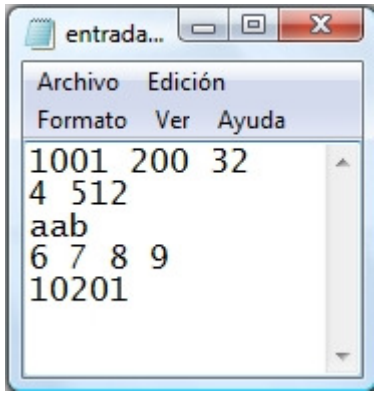
Estructuras de control

Instrucciones de entrada (lectura) y de salida (escritura) (cont.)

- Funciones fscanf y fprintf (cont.)

  format va encerrada entre comillas simples, y contiene los especificadores de formato para las variables

  %s para cadenas de caracteres
  %d para variables enteras
  %f para variables de punto flotante
  %lf para variables de doble precisión



Ejemplos:

```
>> fi1 = fopen('entrada1.txt', 'r');
>> s = fscanf( fi1, '%s');      →   lee una cadena de caracteres

>> fi2 = fopen('entrada1.txt', 'r');
>> s = fscanf( fi2, '%d');      →   lee todos enteros posibles en el archivo

>> fi2 = fopen('entrada1.txt', 'r‘)
>> s = fscanf( fi2, '%d‘,1);    →   lee un entero
```

Estructuras de control

Instrucciones de entrada (lectura) y de salida (escritura) (cont.)

- Funciones fscanf y fprintf (cont.)

  Forma general de la función fprintf:

  count = fprintf(fi, format, var, …)

  Dirige su salida formateada hacia el archivo indicado por el identificador **fi**, **format** contiene los formatos de escritura y **count** retorna el número de bytes escritos satisfactoriamente.

  Ejemplos:

  >> fi1 = fopen(‘salida1’, ‘w’);
  >> count = fprintf(fi1, ‘el número de ecuaciones es %d \n’, n);

  → escribe el texto entre comillas y el valor de la variable n según el formato indicado (número entero)

  >> fi2 = fopen(‘salida2’, ‘w’);
  >> count = fprintf(fi2, ‘el determinante es %10.4f \n’, n);

  Obs. \n en el formato obliga a crear una línea nueva al final del texto.

Estructuras de control

Instrucción de lectura “textread”

Lee datos numéricos (sin formato) o heterogéneos (con formato) de un archivo de texto (ascii)

Sintaxis sin usar formato:

a = textread(archivo)

a = textread(archivo, ‘ ', n)

a = textread(archivo, ‘ ', param, valor, ...)

a = textread(archivo, ‘ ', n, param, valor, ...)

Lee **n** líneas de **archivo** y las almacena en la variable **a**. Se supone que **archivo** contiene sólo datos numéricos.

Opciones para **param**

- 'delimiter': identifica el carácter de separación entre los datos
- 'headerlines': número de líneas de cabecera del archivo, estas líneas son ignoradas en la lectura
- 'commentstyle‘: con valor 'matlab‘, indica que los caracteres después de % son ignorados

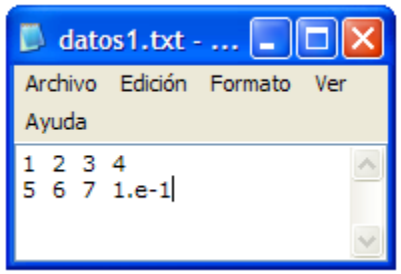
Estructuras de control

Instrucción de lectura “textread” (cont.)

Ejemplos:

>> a = textread('datos1.txt ')

→ lee todas las líneas del archivo

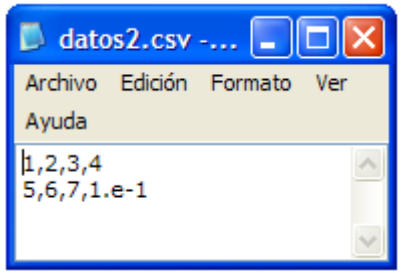


>> a = textread('datos1.txt ', ' ', 1)

→ lee la primera línea del archivo

>> a = textread('datos2.csv ', ' ', 'delimiter', ',')

→ lee todas las líneas del archivo



>> a = textread('datos2.csv', ' ', 1, 'delimiter', ',')

→ lee la primera línea del archivo

>> a = textread('datos22.csv', ' ', 'delimiter', ',', 'headerlines', 1)

→ lee todas las líneas del archivo después del encabezado

>> a = textread('datos23.csv', ' ', 'delimiter', ',', 'headerlines', 1, 'commentstyle', 'matlab') → lee todas las líneas del archivo después del encabezado, ignorando las comentadas

Estructuras de control

Instrucción de lectura “textread” (cont.)

Sintaxis usando formato:

[a, b, c, …] = textread(archivo, formato)

[a, b, c, …] = textread(archivo, formato, n)

[a, b, c, …] = textread(archivo, formato, param, valor, ...)

[a, b, c, …] = textread(archivo, formato, n, param, valor, ...)

Lee **n** líneas de **archivo** con **formato** especificado y las almacena en las variables **a**, **b**, **c**, etc., respectivamente. Los datos en **archivo** pueden ser heterogéneos (números y caracteres), pero se espera que estén organizados homogéneamente por columnas.

- El tipo de dato de cada columna se suministra en **formato** y corresponde a cada variable **a**, **b**, **c**, …
- Si **n** no se especifica o si es -1, lee el archivo completo, en caso contrario lee **n** líneas

Formatos:

%n : números reales o enteros

%d : números enteros

%f : números reales

%s : cadena de caracteres

%5c : 5 caracteres (incluye espacios en blanco)

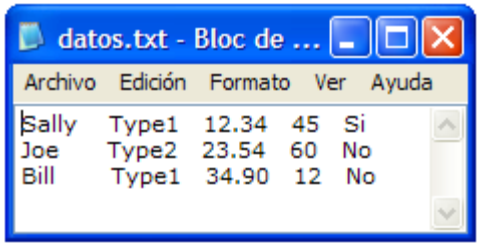
Estructuras de control

Instrucción de lectura “textread” (cont.)

Ejemplos:

```
>> [nombre, tipo, x, y, respuesta] = …
        textread('datos.txt', '%s %s %f %d %s')
```

→ lee todas las líneas del archivo, según el formato especificado y las almacena en las variables indicadas




```
nombre =
    'Sally'
    'Joe'
    'Bill'
tipo =
    'Type1'
    'Type2'
    'Type1'
x =
   12.3400
   23.5400
   34.9000
y =
    45
    60
    12
respuesta =
    'Si'
    'No'
    'No'
```

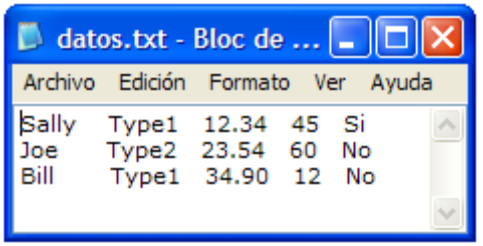
```
>> nombre = textread('datos.txt', '%s %*[^\n]')
```

→ lee la primera columna

```
nombre =
    'Sally'
    'Joe'
    'Bill'
```

Estructuras de control

Instrucciones de lectura “textread” (cont.)




```
>> inicial = textread('datos.txt', '%c %*[^\n]')
```

→ lee el primer caracter

```
inicial =
   S
   J
   B
```

```
>> x = textread('datos.txt', '%*s %*s %f %*d %*s')
```

→ lee la tercera columna

```
x =
   12.3400
   23.5400
   34.9000
```

```
>> a = textread('datos.txt', '%s', 'delimiter', '\n')
```

→ cada línea como cadena de caracteres

```
a =
    'Sally    Type1   12.34   45   Si'
    'Joe      Type2   23.54   60   No'
    'Bill     Type1   34.90   12   No'
```

Estructuras de datos

Arreglos de cadenas de caracteres

Un arreglo de cadena de caracteres es como un vector cuyos elementos son cadenas de caracteres. Este puede ser creado usando llaves, así

```
>> s1 = {‘hola’  ‘si’  ‘adios’}        →    s1 =
                                             'hola'    'si'    'adios'
>> ss =  {'saul' 'pedro' ; 'pepe' 'maria'}
        →    ss =
                 'saul'    'pedro‘
                 'pepe'    'maria'
```

Si construimos la cadena de caracteres concatenados verticalmente

```
>> s2 = strvcat(‘hola', 'si', ‘adios')
```

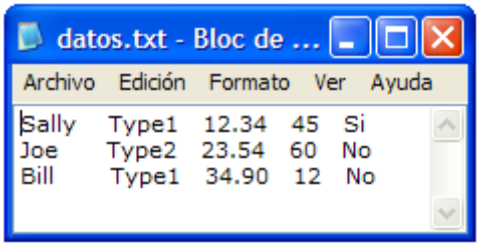
La podemos interpretar como “arreglo de cadena de caracteres”, pero es no es tal.

Se tiene que s1 y s2 son diferentes, pero se puede pasar de un tipo al otro usando las funciones “char” y “cellstr”.

Además existen las funciones “ischar” y “iscellstr” para identificar entre cada uno de ellos.

Estructuras de control

Instrucciones de lectura “textread” (cont.)



Dado el arreglo de cadena de caracteres nombre:

```
>> nombre

            nombre =
                'Sally'
                'Joe'
                'Bill'
```

Podemos agregar una cadena de caracteres a este arreglo:

```
>> nombre = cellstr(strvcat(char(nombre),‘Pedro'))
```

→ agrega al arreglo de cadena de caracteres ‘nombre’ el elemento ‘Pedro’

```
            nombre =
                'Sally'
                'Joe'
                'Bill'
                ‘Pedro'
```

```
>> size(nombre)   →   ans = 4    1
```

Ejemplo:

Leer un conjunto de matrices de un archivo, determinar cuales son DD y cuales no, y las escribe en un archivo con el resultado.
Archivo de entrada: datos_matrices_DD.txt
Archivo de salida: salida_matrices_DD.txt
Procedimiento: lectura_matrices_DD.m

Definición:

Una matriz A de orden *n*×*n* es **diagonal dominante estricta** si

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad 1 \leq i \leq n$$

La matriz $A = \begin{pmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{pmatrix}$ es diagonal dominante estricta, ya que

$$|7| > |2| + |0|$$
$$|5| > |3| + |-1|$$
$$|-6| > |0| + |5|$$

Estructuras de control

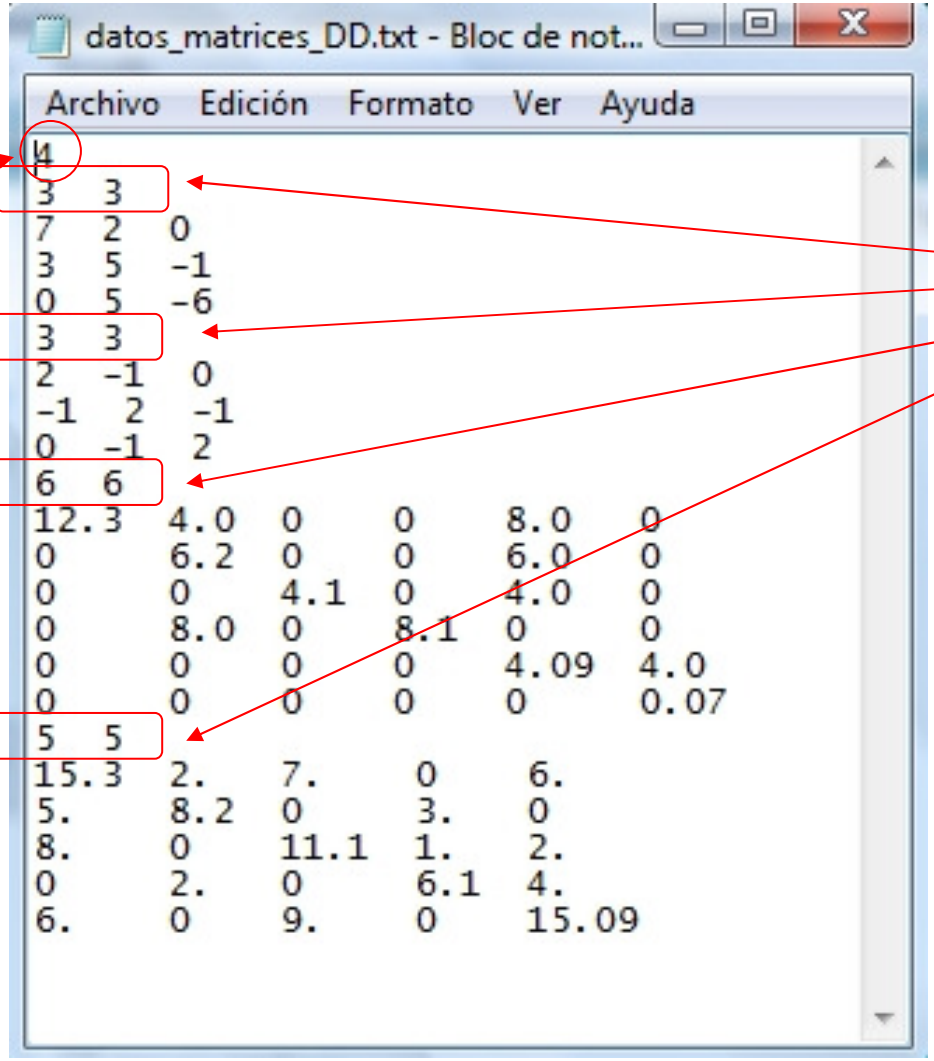
Ejemplo:

Archivo de entrada:

datos_matrices_DD.txt

número de matrices

dimensiones de las matrices



Estructuras de control

Ejemplo:

Archivo de entrada:

datos_matrices_DD.txt

Archivo de salida:

salida_matrices_DD.txt

Procedimiento:

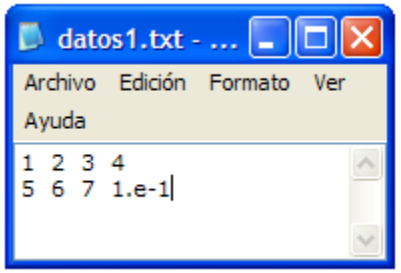
lectura_matrices_DD.m

```
function lectura_matrices_DD(archivo)
% Lee un conjunto de matrices de un archivo y determina cuales
% son DD y cuales no.
% La estructura de datos del archivo de lectura esta dado en
% el archivo datos_matrices.txt
% sintaxis: lectura_matrices(archivo)

f1 = fopen(archivo,'r'); % archivo de entrada
if f1 == -1
    disp('el archivo no existe');
    return
end
f2 = fopen('salida_matices.txt','w'); % archivo de salida
num_matrices = fscanf(f1,'%d',1);
for nn = 1:num_matrices
    num_filas = fscanf(f1,'%d',1);
    num_columnas = fscanf(f1,'%d',1);
    A = zeros(num_filas,num_columnas);
    for i = 1:num_filas
        A(i,:) =fscanf(f1,'%f',num_columnas);
    end
    % escritura de la matriz en el archivo de salida
    for i=1:num_filas
      for j=1:num_columnas
        fprintf(f2,'%f  ',A(i,j));
      end
      fprintf(f2,'%c%c\n',char(13),char(10));
    end
    % determina si la matriz A es diagonal dominante o no
    dom_diag = all(diag(abs(A)) > sum(abs(A-diag(diag(A))),2));
    if dom_diag == 1
        fprintf(f2,'es diagonal dominante \n');
    else
        fprintf(f2,'no es diagonal dominante \n');
    end
    % salto de linea para separacion de los resultados
    % char(13) es "carriage return"
    fprintf(f2,'%c\n%c\n',char(13),char(13));
end
fclose('all');

end
```

Estructuras de control

Instrucción de lectura “load”

Lee el contenido de un “archivo” y lo deposita en un arreglo con nombre “archivo”.



>> load datos1.txt

→ almacena el contenido en la variable **datos1**

la instrucción es equivalente a load –ascii datos1.txt

>> s = load(‘datos1.txt’)

→ almacena el contenido en la variable **s**

>> var = ‘datos1.txt’; s = load(var) → almacena en la variable **s**

Obs:

- El archivo puede tener cualquier extensión diferente a “.mat”, el archivo es tratado como ascii
- Es importante que todas las filas tengan el mismo número de columnas
- Las separaciones entre elementos: blancos o comas (,)
- El archivo debe tener sólo números